# FKME - User Manual

**COLLABORATORS**

| | *TITLE* :<br><br>FKME - User Manual | |
|---|---|---|

| *ACTION* | *NAME* | *DATE* | *SIGNATURE* |
|---|---|---|---|
| WRITTEN BY | limes datentechnik gmbh | Aug 21 2015 | |

**REVISION HISTORY**

| NUMBER | DATE | DESCRIPTION | NAME |
|---|---|---|---|
| 5.1.8 | Aug 21 2015 | released | LDG |

# Contents

**1 FKME overview** **1**

1.1 The use of FKME in FLAM . . . . . 2

**2 FKME FUNCTION FKMESTD0** **3**

**3 FKME FUNCTION SYMNP11** **4**

3.1 ENCRYPTION FUNCTION SYMNP11 *PKCS11* . . . . . 4

3.1.1 PARAMETER *LABEL* . . . . . 5

3.1.2 PARAMETER *TEMPLATE* . . . . . 5

3.2 DECRYPTION FUNCTION SYMNP11 *PKCS11* . . . . . 6

3.2.1 PARAMETER *TEMPLATE* . . . . . 6

**4 FKME FUNCTION SYMNCCA** **7**

4.1 ENCRYPTION FUNCTION SYMNCCA *CCA* . . . . . 7

4.1.1 PARAMETER *LABEL* . . . . . 8

4.1.2 PARAMETER *TEMPLATE* . . . . . 8

4.2 DECRYPTION FUNCTION SYMNCCA *CCA* . . . . . 9

4.2.1 PARAMETER *TEMPLATE* . . . . . 9

**5 FKME FUNCTION SYMSWE** **10**

5.1 ENCRYPTION FUNCTION SWESYM *SWE* . . . . . 10

5.2 DECRYPTION FUNCTION SWESYM *SWE* . . . . . 10

**6 FKME FUNCTION FKMEFILE** **12**

6.1 ENCRYPTION/DECRYPTION FUNCTION FKMEFILE *FIL* . . . . . 12

**7 FKME FUNCTION FKMEFILE** **13**

**8 Index** **14**

Frankenstein Limes Key Management Extension (FKME) Copyright © limes datentechnik ® gmbh All rights reserved

**Trademarks** Below, you can find all trademarks or registered trademarks of limes datentechnik ® gmbh. These trademarked terms are marked with the appropriate symbol (® or ™), indicating registered or common law trademarks owned by limes datentechnik ® gmbh at the time this information was published. The following terms are trademarks of limes datentechnik ® gmbh in Germany, other countries, or both:

- limes® - Short company name of the owner of this document
- limes datentechnik® - Company name of the owner of this document
- FLCL® - Frankenstein Limes Command Line
- FLCC® - Frankenstein Limes Control Center
- FLAM® - Frankenstein Limes Access Method
- FLUC® - Frankenstein Limes Universal Converter
- FLIES® - Frankenstein Limes Integrated Extended Security
- FLAMFILE® - A file based on FLAM syntax

**Abstract** libfkme collects all FKME implementations of limes datentechnik in one library. For each supported specification (FINPIN, KMIP, PGP) several implementations (Software (SWE), PKCS#11 (P11), IBM CCA) are available. This document describes all the currently available FKME functions.

The FLAM Key Management Extension (FKME) is a service provider interface to integrate FLAM into different cryptographic infrastructures.

# PREFACE

The FKME parameter list is parsed based on the CLE/P library. The CLE/P library was developed by limes datentechnik and released as open source under the ZLIB license. CLE/P is a compiler to provide a platform-independent command line interface for each kind of batch processing environment.

The CLE/P library provides a lot of features, including help and documentation. For example: We use this library to automatically create this document as part of our build process. If we add a parameter to the CLE/P tables and build the FL5 project, this manual will be regenerated as well in order to be always up to date.

This manual is generated with the INFO command of FLCL provided by CLE/P library by calling the command below:

```
flcl info get.fkme(docu) output=fkmebook.txt
```

# Chapter 1

# FKME overview

The FLAM key-management extension (FKME) links between FLAM's cryptographic protection mechanisms (privacy, integrity, completeness), various cryptographic infrastructures (KMIP, x509-PKI, FINPIN, ...), and various architectures of hardware security modules (HSM (IBM-CCA/ICSF, PKCS#11, ...)) in order to provide a professional key management by which access to data can be controlled. Of course, FLAM also supports protection by a simple passphrase or an internal constant as keys, but professional solutions are implemented by means of this service-provider interface which has been available since FLAM version 4.

BENEFITS

- Use of existing cryptographic infrastructures for protecting "flambéed" data
- Processes for key management and permission granting can be re-used
- Top security due to support of various hardware security modules (HSM)
- No downstream costs caused by encryption (reuse of key management)
- Compliance with security requirements and standards (PCIDSS)

**Implementation** The following FLAM4-based solutions are currently available:

- FIN/PIN Symmetric Key Infrastructure for data exchange
  - PKCS#11 HSM
  - IBM-CCA-based HSM
  - Software simulation
- FKMESTD0 Default FKME providing the default passphrase
- FKMEFILE Reads the key value (passphrase) from a file

The FIN/PIN implementation for PCIDSS is a specification for secure ordering of debit and credit cards that relies on the existing cryptographic infrastructure for the Financial-PIN-Support. For this, there are meanwhile two different specifications: one for triple-DES and also one for AES which exists in two versions (transfer and storage).

The following FKMEs are planned as part of the FLAM5 project:

- OpenPGP keyrings
- x509 public key infrastructure
- KMIP (Key-Management-Interoperability-Protocol)

In various other projects, customers have developed their own specifications and implemented their own solutions.

## 1.1 The use of FKME in FLAM

This chapter describes the usage of FKMEs developed by limes datentechnik. Calling custom developed FKME works in the same way, but mostly has another structure for the FKME parameter string. The FLAM subsystem on ZOS has special LE-less FKMEs developed in assembler implementing the FIN/PIN specification against ICSF (FKMECCAx). These load modules are still available and must be used with the subsystem. *libfkme* is an LE-based DLL and cannot be used with this environment. The parameter string of FKMECCAx differs from the *libfkme* parameter string of function SYMNCCA, but it does the same thing. Documentation for the FKMECCAx load modules can be found in the FLAM4 user manual.

To call a simple load module on ZOS, only the function name must be defined. To call a function of *libfkme* on ZOS, the library name of the DLL and function name must be defined. On other platforms (Windos, UNIX), the default library name is *libfkme* and the default function name is *FKMESTD0*. *FKMESTD0* is also the default function name on z/OS.

**NOTE** In some environments, you have to escape quotation marks of the FKME parameter string with backslash: \".

# Chapter 2

# FKME FUNCTION FKMESTD0

FKME function FKMESTD0 is an FKME using the default password/key and needs no parameter. This FKME is called if no other function is defined.

**EXAMPLE**

```
...kml=libfkme kmf=FKMESTD0...
```

# Chapter 3

# FKME FUNCTION SYMNP11

FKME function SYMNP11 is an FKME which uses the PKCS#11 secure token interface to implement the FINPIN-based specification for PCIDSS conform data exchange. Currently, it supports two variants: AES (with SHA-256) and 3DES/TDES (with SHA-1).

This FKME function can be used with several PKCS#11 conform crypto devices on Windows / Unix platforms or EP11 on ZOS. The specification was designed for PCIDSS conform ordering of credit cards and to exchange other card holder data in a secure and PCIDSS conform manner.

The specification can be found at:

`http://www.flam.de/en/technology/download/documentation/`

On ZOS, this FKME is available (like on all other platforms) as function of the LIBFKME DLL and also as separate load module *SYMNP110*.

## 3.1 ENCRYPTION FUNCTION SYMNP11 *PKCS11*

**SYNOPSIS**

```
HELP:   FKME PKCS#11 parameter
PATH:   de.limes.
TYPE:   OBJECT
SYNTAX: :>  PKCS11(LIBRARY='str',SLOT=num,PIN='str',ALGO=AES/TDES,LABEL='str',TEMPLATE='str ↩
    ',KEYLENGTH=num/KL16/KL24/KL32)
```

**DESCRIPTION** Parameter of FKME library libfkme function SYMNP11 for encryption.

**EXAMPLE**

```
...kml=libfkme kmf=SYMNP11 kmp="pkcs11(library=p11lib slot=0 pin=1234 ALGO=AES label= ↩
    FKME4711 template=FKME++** KEYLENGTH=16)"...
```

ARGUMENTS

- `STRING:LIBRARY='str' -PKCS#11 library name (DLL/SO)`
- `NUMBER:SLOT=num -Slot number of PKCS#11 token[0]`
- `STRING:PIN='str' -PKCS#11 pin for authentication`
- `NUMBER:ALGO=AES/TDES -Algorithm used by FKME [TDES]`
  - `AES -AES algorithm`

  - TDES -Triple DES (3DES) algorithm

- NUMBER:KEYLENGTH=num/KL16/KL24/KL32 -Key length of FMKY [KL16]
  - KL16 -128 Bit AES /112 Bit TDES
  - KL24 -192 Bit AES /168 Bit TDES
  - KL32 -256 Bit AES

### 3.1.1 PARAMETER *LABEL*

**SYNOPSIS**

```
HELP:   PKCS#11 key label
PATH:   PKCS11
TYPE:   STRING
SYNTAX: LABEL='str'
```

**DESCRIPTION** The fully qualified label is only required to reference the correct key value on write. It must contain a generation and version (2 hex digits each) which are determined based on the key label template.

### 3.1.2 PARAMETER *TEMPLATE*

**SYNOPSIS**

```
HELP:   PKCS#11 key label template
PATH:   PKCS11
TYPE:   STRING
SYNTAX: TEMPLATE='str'
```

**DESCRIPTION** The FKME template is used on write to determine the generation and version from a key label.

The replacement characters below are defined:

```
   Generation '++'
   Version    '**'
```

All other characters of the label can be substituted by %, to define the position of generation and version in the label. The remaining characters must correspond to those in the label.

For example:

```
   TEMPLATE='TFMKY.%%%%%%%%.%%%%%%%%.DAT0++**'
```

On read, only a key label template must be provided. The generation (**) and version (++) is filled in by the FKME. The wildcard % is not allowed.

```
   TEMPLATE='TFMKY.BV000000.GUD00000.DAT0++**'
```

If the label is for example:

```
   LABEL='TFMKY.BV000000.GUD00000.DAT04711'
```

then the generation is 47 and the version is 11.

## 3.2 DECRYPTION FUNCTION SYMNP11 *PKCS11*

**SYNOPSIS**

```
HELP:   FKME PKCS#11 parameter
PATH:   de.limes.
TYPE:   OBJECT
SYNTAX: :>  PKCS11(LIBRARY='str',SLOT=num,PIN='str',TEMPLATE='str')
```

**DESCRIPTION** Parameter of FKME library libfkme function SYMNP11 for decryption.

**EXAMPLE**

```
 ...kml=libfkme kmf=SYMNP11 kmp="pkcs11(library=cryptoky slot=0 pin=1234 template=FKME++**) ↩
    "...
```

ARGUMENTS

- `STRING:LIBRARY='str' -PKCS#11 library name (DLL/SO)`
- `NUMBER:SLOT=num -Slot number of PKCS#11 token[0]`
- `STRING:PIN='str' -PKCS#11 pin for authentication`

### 3.2.1 PARAMETER *TEMPLATE*

**SYNOPSIS**

```
HELP:   PKCS#11 key label template
PATH:   PKCS11
TYPE:   STRING
SYNTAX: TEMPLATE='str'
```

**DESCRIPTION** The FKME template is used on write to determine the generation and version from a key label.

The replacement characters below are defined:

```
   Generation '++'
   Version    '**'
```

All other characters of the label can be substituted by %, to define the position of generation and version in the label. The remaining characters must correspond to those in the label.

For example:

```
   TEMPLATE='TFMKY.%%%%%%%%.%%%%%%%%.DAT0++**'
```

On read, only a key label template must be provided. The generation (**) and version (++) is filled in by the FKME. The wildcard % is not allowed.

```
   TEMPLATE='TFMKY.BV000000.GUD00000.DAT0++**'
```

If the label is for example:

```
   LABEL='TFMKY.BV000000.GUD00000.DAT04711'
```

then the generation is 47 and the version is 11.

# Chapter 4

# FKME FUNCTION SYMNCCA

FKME function SYMNCCA, is an FKME which uses IBM CCA (Common Cryptographic Architecture (ICSF on ZOS)) to implement the FINPIN based specification for PCIDSS conform data exchange. Currently, it supports two variants: AES (with SHA-256) and 3DES/TDES (with SHA-1).

This FKME function can be used with ICSF on ZOS or IBM47xx Cryptocards on Windows or UNIX platforms. The specification was designed for PCIDSS conform ordering of credit cards and to exchange other card holder data in a secure and PCIDSS conform manner.

The specification can be found at URL below:

`http://www.flam.de/en/technology/download/documentation/`

For ICSF, the library to find the callable services must not be specified because all functions are simple load modules in the dataset *CSF.SCSFMOD0*. These modules are fetched by *libfkme*. If you would specify a library name, *libfkme* would try to load a ZOS-DLL, which would not work. To fetch the ICSF service routines, the *CSF.SCSFMOD0* library must be in the STEPLIB concatenation for the program.

The default library name for SAPI on windows is *csunsapi* and on UNIX systems *libcsulsapi*. The directory for this DLL/SO must be defined in the library path environment variable.

On ICSF-based CCA systems (z/OS), authentication against the CCA HSM is not possible. For such an environment, please don't use the userid and passphrase login. On other systems, the userid and passphrase are optional. If you don't provide them, the default role is used.

On ZOS, this FKME is available (like on all other platforms) as a function of the LIBFKME DLL and additionally as separate load module *SYMNCCA0*.

## 4.1 ENCRYPTION FUNCTION SYMNCCA *CCA*

**SYNOPSIS**

```
HELP:   FKME CCA/ICSF parameter
PATH:   de.limes.
TYPE:   OBJECT
SYNTAX: :>  CCA(LIBRARY='str',USERID='str',PASSWORD='str',ALGO=AES/TDES,LABEL='str', ↩
    TEMPLATE='str',KEYLENGTH=num/KL16/KL24/KL32)
```

**DESCRIPTION** Parameter of FKME library libfkme function SYMNCCA for encryption.

**EXAMPLE**

```
...kml=libfkme kmf=SYMNCCA kmp="cca(library=csunsapi user=smith password=1234 ALGO=AES  ↩
    label=FKME4711 template=FKME++** KEYLENGTH=16)"...
```

ARGUMENTS

- STRING:LIBRARY='str' -CCA library name (DLL/SO)
- STRING:USERID='str' -User ID for authentication [optional]
- STRING:PASSWORD='str' -Password for authentication [optional]
- NUMBER:ALGO=AES/TDES -Algorithm used by FKME [TDES]
  - AES -AES algorithm
  - TDES -Triple DES (3DES) algorithm
- NUMBER:KEYLENGTH=num/KL16/KL24/KL32 -Key length of FMKY [KL16]
  - KL16 -128 Bit AES /112 Bit TDES
  - KL24 -192 Bit AES /168 Bit TDES
  - KL32 -256 Bit AES

### 4.1.1 PARAMETER *LABEL*

**SYNOPSIS**

```
HELP:   CCA key label
PATH:   CCA
TYPE:   STRING
SYNTAX: LABEL='str'
```

**DESCRIPTION** The fully qualified label is only required to reference the correct key value on write. It must contain a generation and version (2 hex digits each) which are determined based on the key label template.

### 4.1.2 PARAMETER *TEMPLATE*

**SYNOPSIS**

```
HELP:   CCA key label template
PATH:   CCA
TYPE:   STRING
SYNTAX: TEMPLATE='str'
```

**DESCRIPTION** The FKME template is used on write to determine the generation and version from a key label.

The replacement characters below are defined:

```
Generation '++'
Version    '**'
```

All other characters of the label can be substituted by %, to define the position of generation and version in the label. The remaining characters must correspond to those in the label.

For example:

```
TEMPLATE='TFMKY.%%%%%%%%.%%%%%%%%.DAT0++**'
```

On read, only a key label template must be provided. The generation (**) and version (++) is filled in by the FKME. The wildcard % is not allowed.

```
TEMPLATE='TFMKY.BV000000.GUD00000.DAT0++**'
```

If the label is for example:

```
LABEL='TFMKY.BV000000.GUD00000.DAT04711'
```

then the generation is 47 and the version is 11.

## 4.2 DECRYPTION FUNCTION SYMNCCA *CCA*

**SYNOPSIS**

```
HELP:   FKME CCA/ICSF parameter
PATH:   de.limes.
TYPE:   OBJECT
SYNTAX: :>  CCA(LIBRARY='str',USERID='str',PASSWORD='str',TEMPLATE='str')
```

**DESCRIPTION** Parameter of FKME library libfkme function SYMNCCA for decryption.

**EXAMPLE**

```
...kml=libfkme kmf=SYMNCCA kmp="cca(library=libcsufsapi user=smith password=1234 template= ↩
    FKME++**)"...
```

ARGUMENTS

- `STRING:LIBRARY='str' -CCA library name (DLL/SO)`
- `STRING:USERID='str' -User ID for authentication [optional]`
- `STRING:PASSWORD='str' -Password for authentication [optional]`

### 4.2.1 PARAMETER *TEMPLATE*

**SYNOPSIS**

```
HELP:   CCA key label template
PATH:   CCA
TYPE:   STRING
SYNTAX: TEMPLATE='str'
```

**DESCRIPTION** The FKME template is used on write to determine the generation and version from a key label.

The replacement characters below are defined:

```
   Generation '++'
   Version    '**'
```

All other characters of the label can be substituted by %, to define the position of generation and version in the label. The remaining characters must correspond to those in the label.

For example:

```
   TEMPLATE='TFMKY.%%%%%%%%.%%%%%%%%.DAT0++**'
```

On read, only a key label template must be provided. The generation (**) and version (++) is filled in by the FKME. The wildcard % is not allowed.

```
   TEMPLATE='TFMKY.BV000000.GUD00000.DAT0++**'
```

If the label is for example:

```
   LABEL='TFMKY.BV000000.GUD00000.DAT04711'
```

then the generation is 47 and the version is 11.

# Chapter 5

# FKME FUNCTION SYMSWE

FKME function SYMSWE is an FKME which simulates the symmetric PKCS#11 or CCA implementations. You can define the secret key as clear value. Currently it supports two variants: AES (with SHA-256) and 3DES/TDES (with SHA-1).

**ATTENTION:** This FKME function extists only for testing purposes. Don't use clear key values in production.

On ZOS, this FKME is available (like on all other platforms) as function of the LIBFKME DLL and also as separate load module *FKMESWE0*.

## 5.1 ENCRYPTION FUNCTION SWESYM *SWE*

**SYNOPSIS**

```
HELP:   FKME software emulation parameter
PATH:   de.limes.
TYPE:   OBJECT
SYNTAX: :>  SWE(ALGO=AES/TDES,KEY='str',GENERATION='str',VERSION='str')
```

**DESCRIPTION** Parameter of FKME library libfkme function SYMSWE for encryption.

**EXAMPLE**

```
...kml=libfkme kmf=SYMSWE kmp="SWE(ALGO=AES KEY=0123456789abcdeffedcba9876543210)"...
```

ARGUMENTS

- NUMBER:ALGO=AES/TDES -Algorithm used by FKME [TDES]
  - AES -AES algorithm
  - TDES -Triple DES (3DES) algorithm
- STRING:KEY='str' -Clear secret key vale in HEX
- STRING:GENERATION='str' -Key generation
- STRING:VERSION='str' -Key version

## 5.2 DECRYPTION FUNCTION SWESYM *SWE*

**SYNOPSIS**

```
HELP:   FKME software emulation parameter
PATH:   de.limes.
TYPE:   OBJECT
SYNTAX: :>  SWE(KEY='str')
```

**DESCRIPTION** Parameter of FKME library libfkme function SYMSWE for decryption.

**EXAMPLE**

```
...kml=libfkme kmf=SYMSWE kmp="SWE(ALGO=TDES KEY=0123456789abcdeffedcba9876543210)"...
```

ARGUMENTS

- `STRING:KEY='str' -Clear secret key value in HEX`

# Chapter 6

# FKME FUNCTION FKMEFILE

The function FKMEFILE is an FKME which reads the secret FLAM® key from a file. This is useful if the secret key should not be printed to the logged output.

The file should only contain the key in its first record.

On ZOS, this FKME is available (like on all other platforms) as function of the LIBFKME DLL and also as separate LE-less load module *FKMEFILE*.

**EXAMPLE**

```
...kml=libfkme kmf=FKMEFILE kmp=filename
```

## 6.1 ENCRYPTION/DECRYPTION FUNCTION FKMEFILE *FIL*

**SYNOPSIS**

```
HELP:   FKME read password from file
PATH:   de.limes.
TYPE:   OBJECT
SYNTAX: :>  FIL(FILENAME='str')
```

# Chapter 7

# FKME FUNCTION FKMEFILE

**DESCRIPTION** The function FKMEFILE is an FKME which reads the secret FLAM® key from a file. This is useful if the secret key should not be printed to the logged output.

The file should only contain the key in its first record.

On ZOS, this FKME is available (like on all other platforms) as function of the LIBFKME DLL and also as separate LE-less load module *FKMEFILE*.

**EXAMPLE**

```
...kml=libfkme kmf=FKMEFILE kmp=filename
```

ARGUMENTS

- `STRING:FILENAME='str' -File with key`

# Chapter 8

# Index

**A**
Argument LABEL, 5, 8
Argument TEMPLATE, 5, 6, 8, 9

**D**
DECRYPTION FUNCTION SWESYM SWE, 11
DECRYPTION FUNCTION SYMNCCA CCA, 9
DECRYPTION FUNCTION SYMNP11 PKCS11, 6

**E**
ENCRYPTION FUNCTION SWESYM SWE, 10
ENCRYPTION FUNCTION SYMNCCA CCA, 8
ENCRYPTION FUNCTION SYMNP11 PKCS11, 4
ENCRYPTION/DECRYPTION FUNCTION FKMEFILE FIL, 13

# COLOPHON

limes datentechnik(R) gmbh
Louisenstrasse 21
D-61348 Bad Homburg v.d.H.
phone: +49(0)6172-5919-0
fax:   +49(0)6172-5919-39
mail:  info@flam.de
web:   www.flam.de or www.limes.de
Amtsgericht: Bad Homburg vor der Hoehe HRB 3288 (gegr. 1985)
Geschaeftsfuehrer: Diplom-Mathematiker Heinz-Ulrich Wiebach
limes datentechnik(R): efficiency at the limit of possibility.